02
Static Array

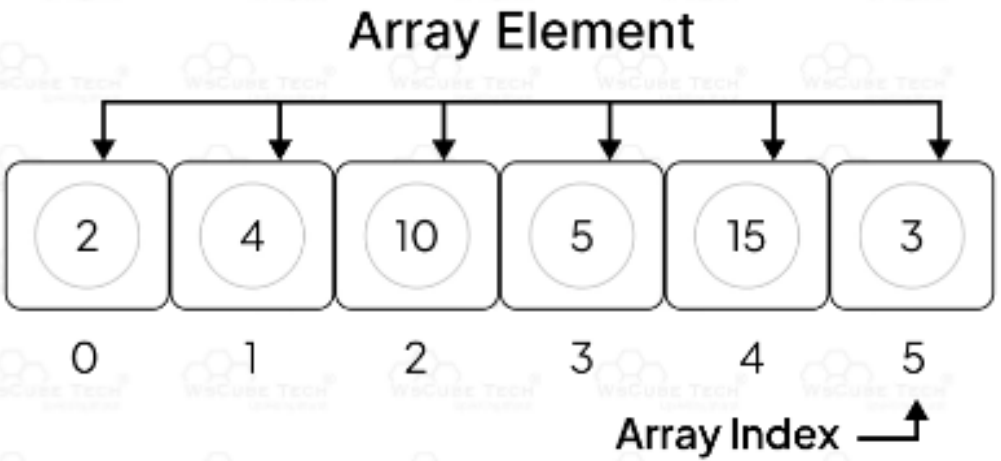
Dr. Mahmoud Alnamoly

Dr. Mahmoud Alna

Static Array ?

An array is a basic data structure used to store a **fixed-size** collection of elements of the **same type**.

These elements are arranged in **contiguous memory locations**, allowing each element to be indexed or accessed directly using an integer index.



Data structure arrays are essential because they form the basis for many other data structures, such as stacks, queues,Tree and array-based lists.

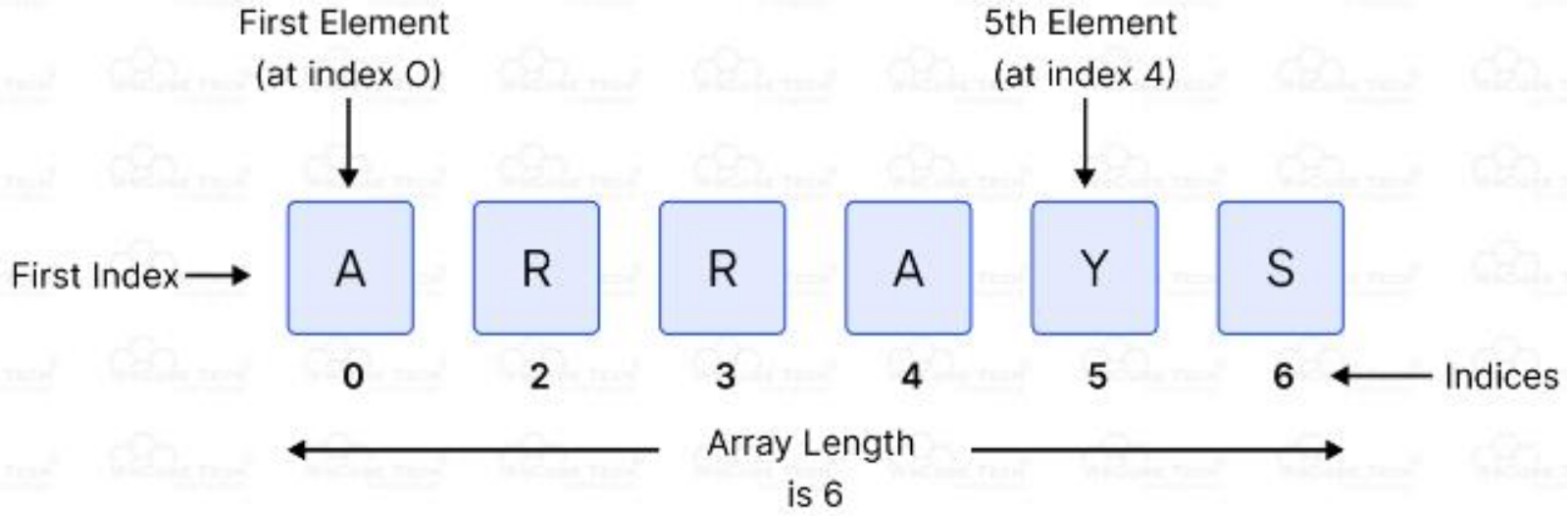
Static Array Characteristics

- ✓ **Fixed Size:** Once an array is declared, its size cannot be changed during runtime. You need to know the number of elements you will store in the array beforehand. Memory is allocated on the **stack**. The allocation is done at **compile time**, and the size is fixed.
- ✓ **Homogeneous Elements:** All elements in an array must be of the same data type, such as all integers, all floats, or all characters.
- ✓ **Indexed by Integers:** Each element in an array is assigned a unique integer called an index, which identifies its position within the array. Indexing usually starts at 0.
- ✓ **Efficient Access:** Because of the way arrays are stored in memory (contiguously), accessing any element by its index is very efficient. This direct access using the index is often referred to as "random access."

Different types of arrays

One-Dimensional Arrays (1D Arrays)

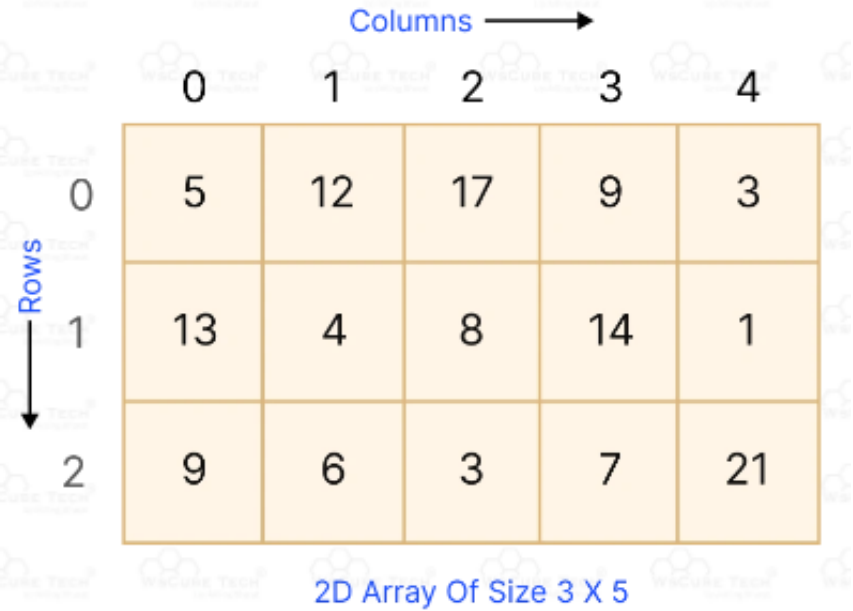
These are the simplest form of arrays, consisting of a single line of elements. Each element in a 1D array is accessed by a single index representing its position within the array.



Different types of arrays

Two-Dimensional Arrays (2D Arrays)

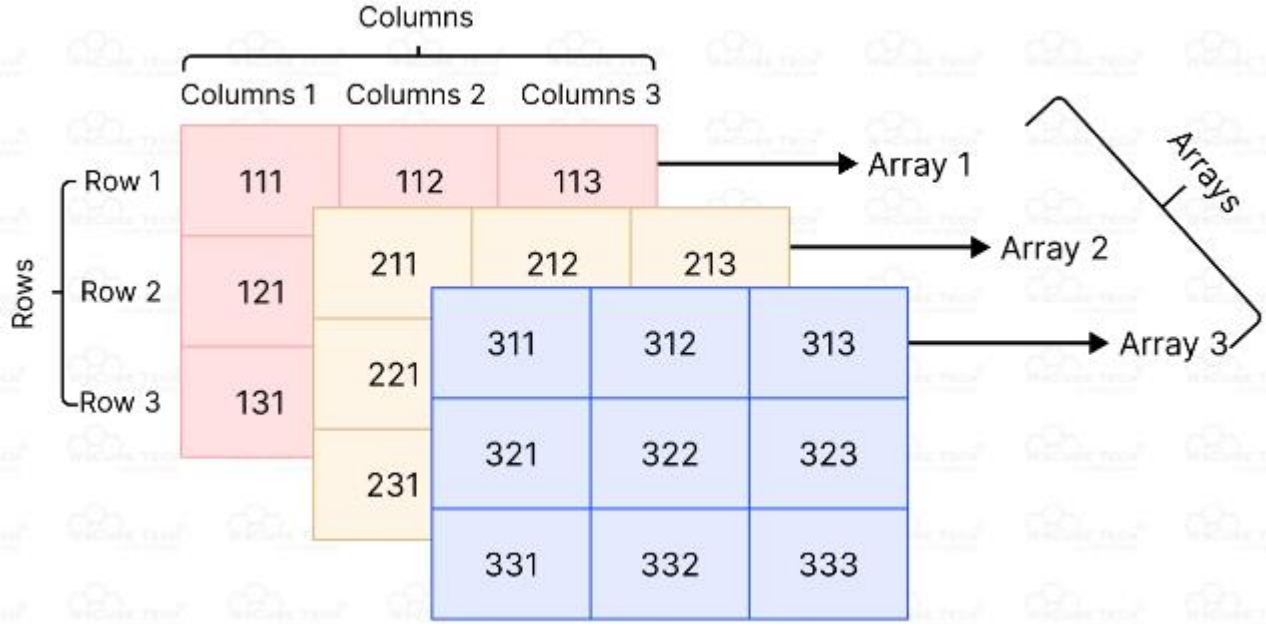
A two-dimensional array in data structure, often thought of as a **matrix**, consists of rows and columns. It is used to represent data in a grid format and is accessed by **two indices**: one for the row and another for the column.



Different types of arrays

Multi-Dimensional Arrays

These are the arrays with more than two dimensions. The multi-dimensional arrays in data structures can be used in scenarios requiring **complex data representation**.



Array Declaration

```
// Declaring 1D static array of integers with size 5
   int numbers[5] = { 1, 2, 3, 4, 5 };
```

```
// Declaring and initializing a 2D array
   int arr[3][3] = {
       {1, 2, 3},
       {4, 5, 6},
       {7, 8, 9}
   };
```

Different Array Operations

Arrays support a variety of operations that allow for the manipulation and interaction with the data they hold:

- ✓ **Access Elements in Array**
- ✓ **Insert Element in Array**
- ✓ **Delete an Element From Array**
- ✓ **Search for an Element in Array**
- ✓ **Update Value of Existing Element in Array**
- ✓ **Sort Array Elements**

1. Access Elements in 1D Array

```cpp
#include <iostream>
using namespace std;

int main() {
    // Declaring a static array of integers with size 5
    int numbers[5] = { 1, 2, 3, 4, 5 };

    // Accessing elements of the array
    for (int i = 0; i < 5; i++) {
        cout << numbers[i] << " ";
    }

    return 0;
}
```

1. Access Elements in 2D Array

```cpp
#include <iostream>
using namespace std;

int main() {
    // Declaring and initializing a 2D array
    int arr[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    // Accessing and printing the 2D array elements
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

2. Insert Elements in 1D Array

```cpp
#include <iostream>
using namespace std;

int main() {
    // Declaring and initializing a 1D array
    int arr[3] = {};

    // Accessing and printing the 1D array elements

    for (int j = 0; j < 3; j++) {
        cout << "arr[" << j << "]= ";
        cin >> arr[j];
    }

    return 0;
}
```

2. Insert Elements in 2D Array

```cpp
#include <iostream>
using namespace std;

int main() {
    // Declaring and initializing a 2D array
    int arr[3][3] = {};

    // Accessing and printing the 2D array elements
    for (int i = 0; i < 3; i++) {
        cout << "\nElements of Row #" << i + 1 << endl;
        for (int j = 0; j < 3; j++) {
            cout << "arr[" << i << "][" << j << "]= ";
            cin >> arr[i][j];
        }
    }

    return 0;
}
```

Microsoft Visual Studio Debug Console

```
Elements of Row #1
arr[0][0]= 5
arr[0][1]= 3
arr[0][2]= 4

Elements of Row #2
arr[1][0]= 8
arr[1][1]= 2
arr[1][2]= 1

Elements of Row #3
arr[2][0]= 3
arr[2][1]= 5
arr[2][2]= 7

C:\Users\MahmoudAlnamoly\source
```

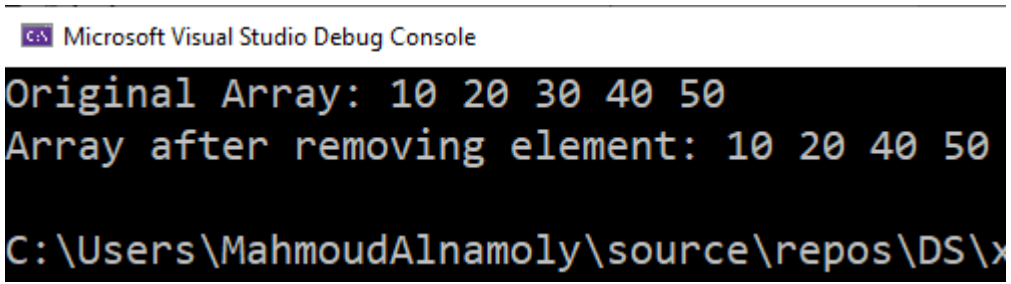
Dr. Mahmoud Alnamoly

3. Delete Elements from Array

```cpp
#include <iostream>
using namespace std;
void removeElement(int arr[], int& n, int index);
int main() {
    int arr[5] = { 10, 20, 30, 40, 50 };
    int n = 5; // Current size of the array

    cout << "Original Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    // Remove element at index 2 (value 30)
    removeElement(arr, n, 2);

    cout << "Array after removing element: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

3. Delete Elements from Array

```
void removeElement(int arr[], int& n, int index) {
    // Check if index is valid
    if (index < 0 || index >= n) {
        cout << "Invalid index" << endl;
        return;
    }

    // Shift elements to the left
    for (int i = index; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }

    // Reduce the size of the array
    n--;
}
```


Microsoft Visual Studio Debug Console
Original Array: 10 20 30 40 50
Array after removing element: 10 20 40 50
C:\Users\MahmoudAlnamoly\source\repos\DS\x

5. Sort Elements in Array

```cpp
#include <iostream>
#include <algorithm>  // Needed for std::sort
using namespace std;

int main() {
    int arr[5] = { 40, 10, 50, 30, 20 };  // Declare and initialize array

    // Sort the array in ascending order
    sort(arr, arr + 5);  // `arr` is the beginning, `arr + 5` is the end of the array

    // Sort in descending order
    sort(arr, arr + 5, greater<int>());
    // Print the sorted array
    cout << "Sorted 1D array: ";
    for (int i = 0; i < 5; i++) {
        cout << arr[i] << " ";
    }

    return 0;
}
```

The time complexity of std::sort is **O(n log n),** which is very efficient.

4. Search Elements in 1D Array

```cpp
#include <iostream>
using namespace std;

// Function to perform a linear search
int linearSearch(int arr[], int n, int target) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            return i;  // Return the index if found
        }
    }
    return -1;  // Return -1 if not found
}
```

4. Search Elements in 1D Array

```
int binarySearch(int arr[], int low, int high, int target) {
    while (low <= high) {
        int mid = low + (high - low) / 2;

        // Check if the target is at the middle
        if (arr[mid] == target) {
            return mid;
        }
        // If target is greater, ignore the left half
        if (arr[mid] < target) {
            low = mid + 1;
        }
        // If target is smaller, ignore the right half
        else {
            high = mid - 1;
        }
    }

    return -1;  // Return -1 if not found
}
```

4. Search Elements in 1D Array

```
int main() {
    int arr[] = { 12, 34, 54, 2, 99 };  // Example array
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 54;

     // Call one of the two search algorithm linear  or binary search
    int result = linearSearch(arr, n, target);
    int result = binarySearch(arr, 0, n - 1, target);

    if (result != -1) {
        cout << "Element found at index " << result << endl;
    }
    else {
        cout << "Element not found" << endl;
    }

    return 0;
}
```

4. Search Elements in 1D Array

Note:-

Linear search is a straightforward method that works for any array but is less efficient with large datasets.

Binary search is more efficient but requires the array to be sorted.

Search Method	Time Complexity	Works on Sorted Arrays?	Works on Unsorted Arrays?
Linear Search	O(n)	Yes	Yes
Binary Search	O(log n)	Yes	No

Array Operations Complexity

Operation	Time Complexity
Access	O(1)
Search	O(n)
Insertion	O(n)
Deletion	O(n)
Updating	O(1)
Sorting	O(n log n) **Merge Sort**

Array Abstract data type (ADT)

Array ADT	
Array is a data structure that is used to store multiple values of similar data types in a contiguous memory location using the following operations.	
Append	Add an element at the end
Insert	Insert an element at specific location
Remove	Remove an element from the end
Delete	Delete an element from specific location
Get	Access/Return an element at specific index
GetSize	Return the current size of the array
Reverse	Reverse array elements
Search	Look up an element on array, return its index
Display	Display array elements
Sort	Sort array elements in ascending and descending order

Based on this ADT we can implement it using class.

Encapsulate all data items and functions (operations) in a class

Then, we can use this class as a data structure

Let's see how it done

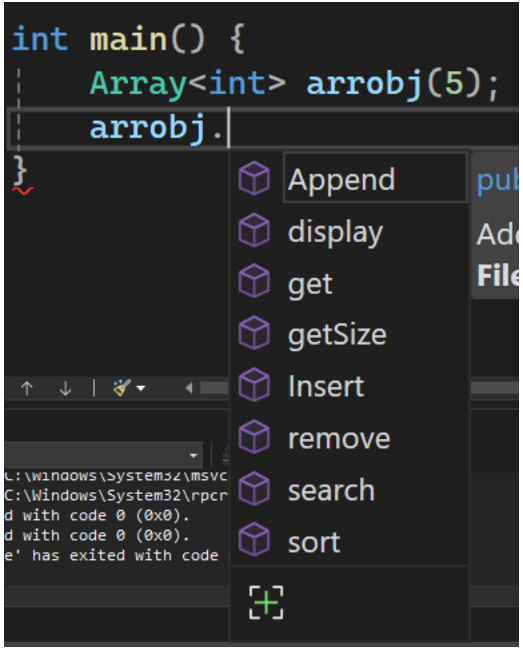
Array Abstract data type (ADT)

Array ADT	
Array is a data structure that is used to store multiple values of similar data types in a contiguous memory location using the following operations.	
Append	Add an element at the end
Insert	Insert an element at specific location
Remove	Remove an element from the end
Delete	Delete an element from specific location
Get	Access/Return an element at specific index
GetSize	Return the current size of the array
Reverse	Reverse array elements
Search	Look up an element on array, return its index
Display	Display array elements
Sort	Sort array elements in ascending and descending order

```cpp
#include <iostream>
#include <stdexcept>

template<class T>
class Array {
private:
    size_t capacity;  // Total Size of array
    size_t size;      // No.of elements in array
    T arr[];
public:
    // Constructor
    Array(size_t cap) :capacity(cap), size(0) {

    }
    // Add element
    void Append(T element) {

    }

    //Insert at specific location
    void Insert(T element, int index) {

    }
```

```
    // Remove element at index
    void remove(size_t index) {

    }

    // Access element at index
    int get(size_t index) const {

    }

    // Search
    T search() const{

    }

    // Sort
    void sort() {

    }
    // Get current size
    size_t getSize() const {

    }
```

```
    // Display array elements
    void display() const {

    }
};

int main() {
    // Create an array with a capacity of 5
    Array<int> arrobj(5);
    arrobj.
}
```



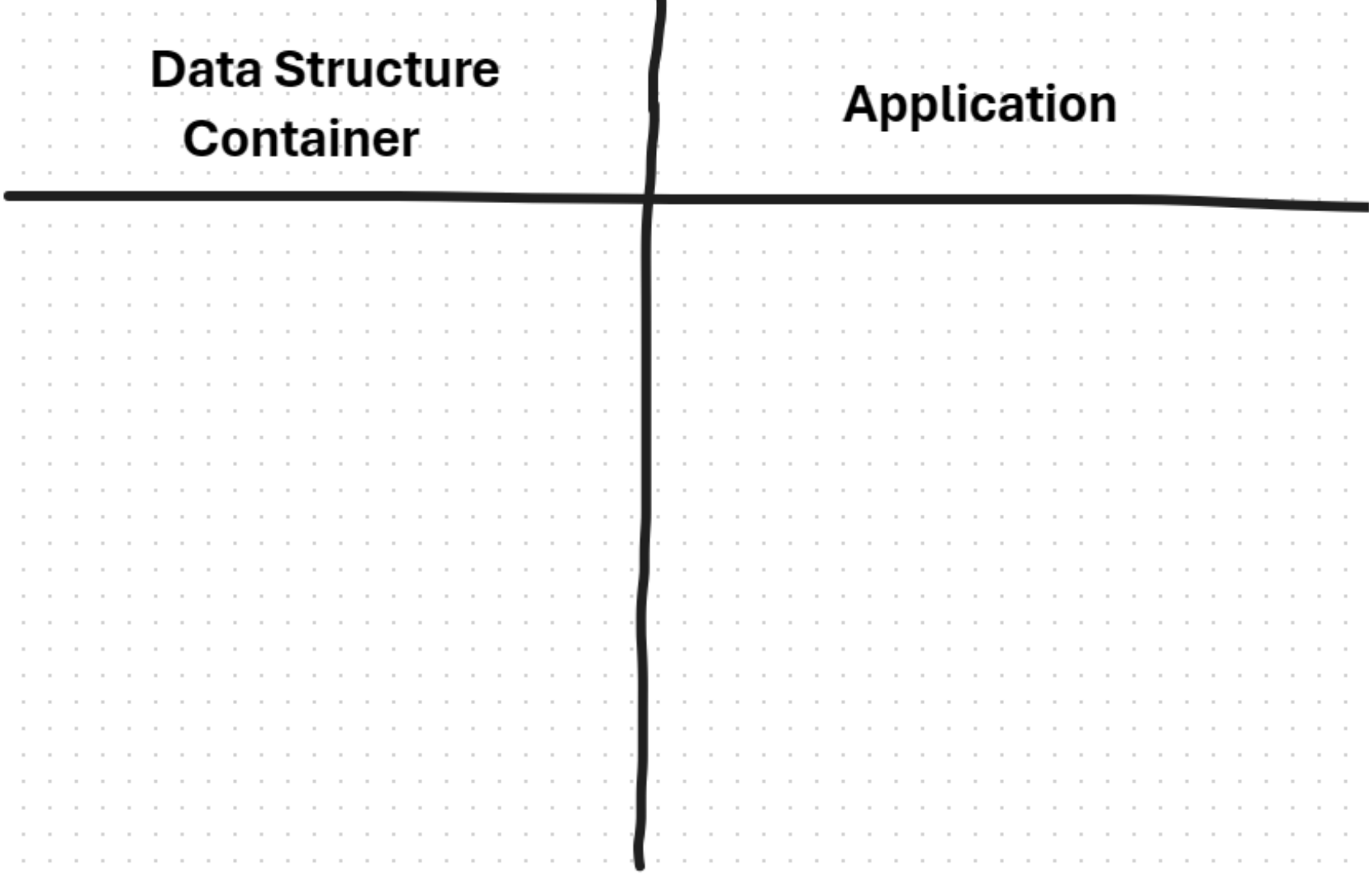
Remove Duplicates from array (+1)

```
int arr[] = { 1, 2, 2, 3, 3, 4, 5, 5 };

int removeDuplicates() {
    if (size == 0) return 0;

    int newSize = 1;
    for (int i = 1; i < size; ++i) {
        if (arr[i] != arr[newSize - 1]) {
            arr[newSize++] = arr[i];
        }
    }
    return newSize;
}
```

```
int newSize = removeDuplicates();

std::cout << "Array after removing duplicates: ";
for (int i = 0; i < newSize; ++i) {
    std::cout << arr[i] << " ";
}
```

Dynamic Array

>>> Next Lecture



Dr.

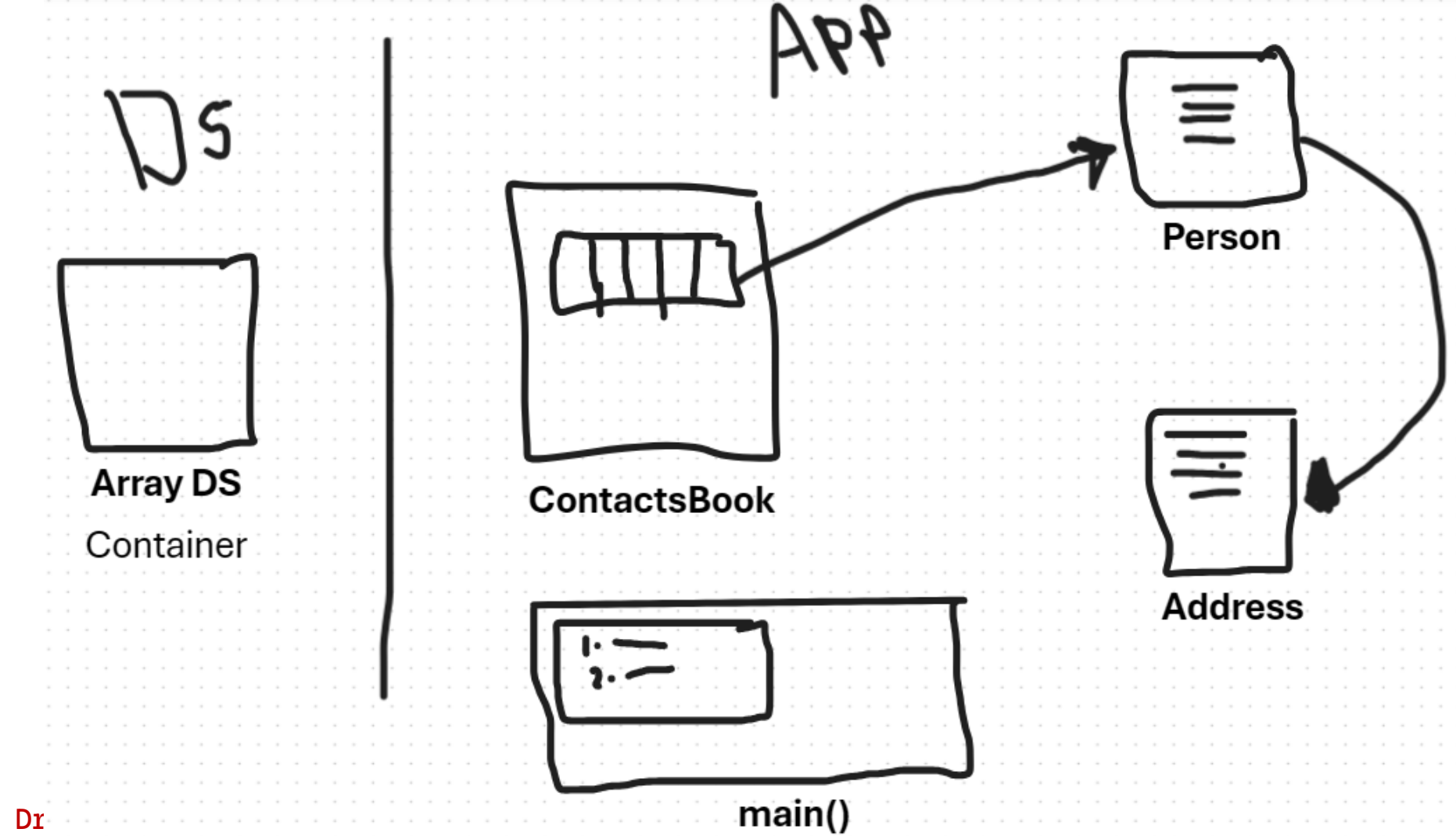
1st Project

CodeWithArrayDS

Deadline

20/10/2024 at 12:00 Am

Dr. Mahmoud Alnamoly


DS
App
Array DS
Container
ContactsBook
Person
Address
main()

Requirements

1. UML Class Diagram of project (Relationships)

2. Container (Data Structure) as a separated file (.h)

3. Final Program tested without any error

Dr. Mahmoud Alna